

A Time-Efficient Output-Sensitive Quantum Algorithm for Boolean Matrix Multiplication

François Le Gall

Department of Computer Science
Graduate School of Information Science and Technology
The University of Tokyo
legall@is.s.u-tokyo.ac.jp

Abstract

This paper presents a quantum algorithm that computes the product of two $n \times n$ Boolean matrices in $\tilde{O}(n\sqrt{\ell} + \ell\sqrt{n})$ time, where ℓ is the number of non-zero entries in the product. This improves the previous output-sensitive quantum algorithms for Boolean matrix multiplication in the time complexity setting by Buhrman and Špalek (SODA'06) and Le Gall (SODA'12). We also show that our approach cannot be further improved unless a breakthrough is made: we prove that any significant improvement would imply the existence of an algorithm based on quantum search that multiplies two $n \times n$ Boolean matrices in $O(n^{5/2-\varepsilon})$ time, for some constant $\varepsilon > 0$.

1 Introduction

1.1 Background

Multiplying two Boolean matrices, where addition is interpreted as a logical OR and multiplication as a logical AND, is a fundamental problem that has found applications in many areas of computer science (for instance, computing the transitive closure of a graph [7, 8, 15] or solving all-pairs path problems [5, 9, 17, 18]). The product of two $n \times n$ Boolean matrices can be trivially computed in time $O(n^3)$. The best known algorithm is obtained by seeing the input matrices as integer matrices, computing the product, and converting the product matrix to a Boolean matrix. Using the algorithm by Coppersmith and Winograd [4] for multiplying integer matrices (and more generally for multiplying matrices over any ring), or its recent improvements by Stothers [19] and Vassilevska Williams [20], this gives a classical algorithm for Boolean matrix multiplication with time complexity $O(n^{2.38})$.

This algebraic approach has nevertheless many disadvantages, the main being that the huge constants involved in the complexities make these algorithms impractical. Indeed, in the classical setting, much attention has focused on algorithms that do not use reductions to matrix multiplication over rings, but instead are based on search or on combinatorial arguments. Such algorithms are often called *combinatorial algorithms*, and the main open problem in this field is to understand whether a $O(n^{3-\varepsilon})$ -time combinatorial algorithm, for some constant $\varepsilon > 0$, exists for Boolean matrix multiplication. Unfortunately, there has been little progress on this question. The best known combinatorial classical algorithm for Boolean matrix multiplication, by Bansal and Williams [2], has time complexity $O(n^3 / \log^{2.25}(n))$.

In the quantum setting, there exists a straightforward $\tilde{O}(n^{5/2})$ -time¹ algorithm that computes the product of two $n \times n$ Boolean matrices A and B : for each pair of indexes $i, j \in \{1, 2, \dots, n\}$,

¹In this paper the notation \tilde{O} suppresses $\text{poly}(\log n)$ factors.

check if there exists an index $k \in \{1, \dots, n\}$ such that $A[i, k] = B[k, j] = 1$ in time $\tilde{O}(\sqrt{n})$ using Grover's quantum search algorithm [10]. Buhrman and Špalek [3] observed that a similar approach leads to a quantum algorithm that computes the product AB in $\tilde{O}(n^{3/2}\sqrt{\ell})$ time, where ℓ denotes the number of non-zero entries in AB . Since the parameter $\ell \in \{0, \dots, n^2\}$ represents the sparsity of the output matrix, such an algorithm will be referred as *output-sensitive*. Classical output-sensitive algorithms for Boolean matrix multiplication have also been constructed recently: Amossen and Pagh [1] constructed an algorithm with time complexity $\tilde{O}(n^{1.724}\ell^{0.408} + n^{4/3}\ell^{2/3} + n^2)$, while Lingas [14] constructed an algorithm with time complexity $\tilde{O}(n^2\ell^{0.188})$. The above $\tilde{O}(n^{3/2}\sqrt{\ell})$ -time quantum algorithm beats both of them when $\ell \leq n^{1.602}$. Note that these two classical algorithms are based on the approach by Coppersmith and Winograd [4] and are thus not combinatorial.

Le Gall [13] has recently shown that there exists an output-sensitive quantum algorithm that computes the product of two $n \times n$ Boolean matrices with time complexity $O(n^{3/2})$ if $1 \leq \ell \leq n^{2/3}$ and $O(n\ell^{3/4})$ if $n^{2/3} \leq \ell \leq n^2$. This algorithm, which improves the quantum algorithm by Buhrman and Špalek [3], was constructed by combining ideas from works by Vassilevska Williams and Williams [21] and Lingas [14].

Several developments concerning the quantum query complexity of this problem, where the complexity under consideration is the number of queries to the entries of the input matrices A and B , have also happened. Output-sensitive quantum algorithms for Boolean matrix multiplication in the query complexity setting were first proposed in [21], and then improved in [13]. Very recently, Jeffery, Kothari and Magniez [11] significantly improved those results: they showed that the quantum query complexity of computing the product of two $n \times n$ Boolean matrices with ℓ non-zero entries is $\tilde{O}(n\sqrt{\ell})$, and gave a matching (up to polylogarithmic factors) lower bound $\Omega(n\sqrt{\ell})$. The quantum query complexity of Boolean matrix multiplication may thus be considered as settled.

Can the quantum time complexity of Boolean matrix multiplication can be further improved as well? The most fundamental question is of course whether there exists a quantum algorithm that uses only quantum search or similar techniques with time complexity $O(n^{5/2-\varepsilon})$, for some constant $\varepsilon > 0$, when $\ell \approx n^2$. This question is especially motivated by its apparently deep connection to the design of subcubic-time classical combinatorial algorithms for Boolean matrix multiplication: a $O(n^{5/2-\varepsilon})$ -time quantum algorithm would correspond to an amortized cost of $O(n^{1/2-\varepsilon})$ per entry of the product, which may provides us with a new approach to develop a subcubic-time classical combinatorial algorithm, i.e., an algorithm with amortized cost of $O(n^{1-\varepsilon'})$ per entry of the product. Studying quantum algorithms for Boolean matrix multiplication in the time complexity setting can then, besides its own interest, be considered as a way to gain new insight about the optimal value of the exponent of matrix multiplication in the general case (i.e., for dense output matrices). In comparison, when the output matrix is dense, the classical and the quantum query complexities of matrix multiplication are both trivially equal to $\Theta(n^2)$.

1.2 Statement of our results

In this paper we build on the recent approach by Jeffery, Kothari and Magniez [11] to construct a new time-efficient output-sensitive quantum algorithm for Boolean matrix multiplication. Our main result is stated in the following theorem.

Theorem 1. *There exists a quantum algorithm that computes the product of two $n \times n$ Boolean matrices with time complexity $\tilde{O}(n\sqrt{\ell} + \ell\sqrt{n})$, where ℓ denotes the number of non-zero entries in the product.*

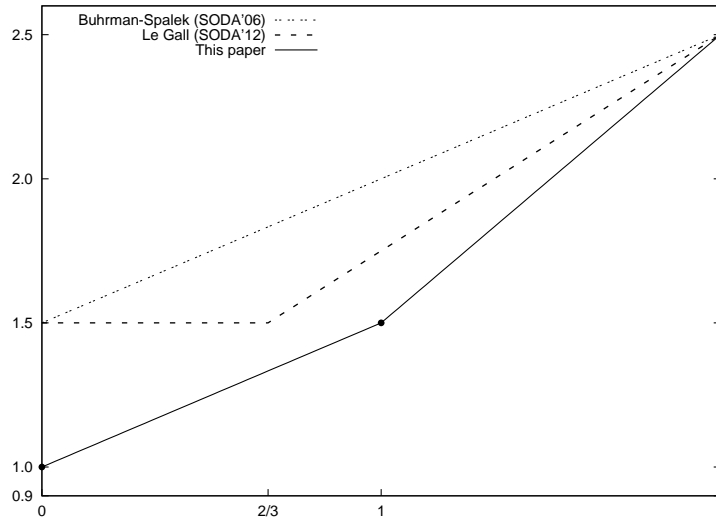


Figure 1: The upper bounds on the time complexity of quantum algorithms for matrix multiplication given in Theorem 1 (in solid line). The horizontal axis represents the logarithm of ℓ with respect to basis n (i.e., the value $\log_n(\ell)$). The vertical axis represents the logarithm of the complexity with respect to basis n . The dashed line represents the upper bounds on the time complexity obtained in [13], and the dotted line represents the upper bounds obtained in [3].

The upper bounds of Theorem 1 are illustrated in Figure 1. Our algorithm improves the quantum algorithm by Le Gall [13] for any value of ℓ other than $\ell \approx n^2$ (we obtain the same upper bound $\tilde{O}(n^{2.5})$ for $\ell \approx n^2$). It also beats the classical algorithms by Amossen and Pagh [1] and Lingas [14] mentioned earlier, which are based on the algebraic approach, for any value $\ell \leq n^{1.847}$ (i.e., whenever $\ell\sqrt{n} \leq n^2\ell^{0.188}$).

As will be explained in more details below, for $\ell \leq n$ our result can be seen as a time-efficient version of the quantum algorithm constructed for the query complexity setting in [11]. The query complexity lower bound $\Omega(n\sqrt{\ell})$ proved in [11] shows that the time complexity of our algorithm is optimal, up to a possible polylogarithmic factor, for $\ell \leq n$. The most interesting part of our results is perhaps the upper bound $\tilde{O}(\ell\sqrt{n})$ we obtain for $\ell \geq n$, which corresponds to the case where the output matrix is reasonably dense and differs from the query complexity upper bounds obtained in [11]. We additionally show that, for values $\ell \geq n$, no quantum algorithm based on search can perform better than ours unless there exists a quantum algorithm based on search that computes the product of two arbitrary $n \times n$ Boolean matrices with time complexity significantly better than $n^{5/2}$. The formal statement follows.

Theorem 2. *Let δ be any function such that $\delta(n) > 0$ for all $n \in \mathbb{N}^+$. Suppose that, for some value $\lambda \geq n$, there exists a quantum algorithm \mathcal{Q} that, given as input any $n \times n$ Boolean matrices A and B such that the number of non-zero entries in the product AB is at most λ , computes AB in $O(\lambda\sqrt{n} \cdot n^{-\delta(n)})$ time. Then there exists an algorithm using \mathcal{Q} as a black-box that computes the product of two $n \times n$ Boolean matrices with overall time complexity $\tilde{O}(n^{5/2-\delta(n)} + n)$.*

The reduction stated in Theorem 2 is actually classical and combinatorial: the whole algorithm uses only classical combinatorial operations and calls to \mathcal{Q} . Thus Theorem 2 implies that, if for a given value $\ell \geq n$ the complexity of Theorem 1 can be improved to $O(\ell\sqrt{n}/n^\varepsilon)$, for some constant $\varepsilon > 0$, using techniques similar to ours (i.e., based on quantum search), then there exists an algorithm based on quantum search (and classical combinatorial operations) that computes the

product of two $n \times n$ Boolean matrices with time complexity $\tilde{O}(n^{5/2-\varepsilon} + n)$.

1.3 Overview of our techniques

The main tool used to obtain our improvements is the new approach by Jeffery, Kothari and Magniez [11] to find collisions in the graph associated with the multiplication of two $n \times n$ Boolean matrices. More precisely, it was shown in [11] how to find up to t collisions in this graph, on a quantum computer, using $\tilde{O}(\sqrt{nt} + \sqrt{\ell})$ queries, where ℓ is the number of non-zero entries in the product. We construct (in Section 3) a time-efficient version of this algorithm that finds one collision in $\tilde{O}(\sqrt{n} + \sqrt{\ell})$ time. We then use this algorithm to design a quantum algorithm that computes the matrix product in time $\tilde{O}(n\sqrt{\ell})$ when $\ell = O(n)$, which proves Theorem 1 for $\ell = O(n)$. Our key technique is the introduction of a small data structure that is still powerful enough to enable time-efficient access to exactly all the information about the graph needed by the quantum searches. More precisely, while the size of the graph considered is $O(n^2)$, we show that the size of this data structure can be kept much smaller — roughly speaking, the idea is to keep a record of the non-edges of the graph. Moreover, the data structure is carefully chosen so that constructing it, at the beginning of the algorithm, can be done very efficiently (in $\tilde{O}(n)$ time), and updating it during the execution of the algorithm can be done at a cost less than the running time of the quantum searches.

We then prove that the ability of finding up to n non-zero entries of the matrix product is enough by showing (in Section 4) a classical reduction, for $\ell > n$, from the problem of computing the product of two $n \times n$ Boolean matrices with at most ℓ non-zero entries in the product to the problem of computing ℓ/n separate products of two Boolean matrices, each product having at most $O(n)$ non-zero entries. The idea is to randomly permute the rows and columns of the input matrices in order to make the output matrix homogeneous (in the sense that the non-zero entries are distributed almost uniformly), in which case we can decompose the input matrices into smaller blocks and ensure that each product of two smaller blocks contains, with non-negligible probability, at most $O(n)$ non-zero entries. This approach is inspired by a technique introduced by Lingas [14] and then generalized in [12, 13]. The main difference is that here we focus on the number of non-zero entries in the product of each pair of blocks, while [12, 13, 14] focused mainly on the size of the blocks. The upper bounds of Theorem 1 for $\ell \geq n$ follow directly from our reduction, and a stronger version of this reduction leads to the proof of Theorem 2.

2 Preliminaries

In this paper we suppose that the reader is familiar with quantum computation, and especially with quantum search and its variants. We present below the model we are considering for accessing the input matrices on a quantum computer, and computing their product. This model is the same as the one used in [3, 13].

Let A and B be two $n \times n$ Boolean matrices, for any positive integer n (the model presented below can be generalized to deal with rectangular matrices in a straightforward way). We suppose that these matrices can be accessed directly by a quantum algorithm. More precisely, we have an oracle O_A that, for any $i, j \in \{1, \dots, n\}$, any $a \in \{0, 1\}$ and any $z \in \{0, 1\}^*$, performs the unitary mapping $O_A: |i\rangle|j\rangle|a\rangle|z\rangle \mapsto |i\rangle|j\rangle|a \oplus A[i, j]\rangle|z\rangle$, where \oplus denotes the bit parity (i.e., the logical XOR). We have a similar oracle O_B for B . Since we are interested in time complexity, we will count all the computational steps of the algorithm and assign a cost of one for each call to O_A or O_B , which corresponds to the cases where quantum access to the inputs A and B can be

done at unit cost, for example in a random access model working in quantum superposition (we refer to [16] for an extensive treatment of such quantum random access memories).

Let $C = AB$ denote the product of the two matrices A and B . Given any indices $i, j \in \{1, \dots, n\}$ such that $C[i, j] = 1$, a witness for this non-zero entry is defined as an index $k \in \{1, \dots, n\}$ such that $A[i, k] = B[k, j] = 1$. We define a quantum algorithm for Boolean matrix multiplication as follows.

Definition 2.1. *A quantum algorithm for Boolean matrix multiplication is a quantum algorithm that, when given access to oracles O_A and O_B corresponding to Boolean matrices A and B , outputs with probability at least $2/3$ all the non-zero entries of the product AB along with one witness for each non-zero entry.*

The complexity of several algorithms in this paper will be stated using an upper bound λ on the number ℓ of non-zero entries in the product AB . The same complexity, up to a logarithmic factor, can actually be obtained even if no nontrivial upper bound on ℓ is known a priori. The idea is, similarly to what was done in [21, 13], to try successively $\lambda = 2$ (and find up to 2 non-zero entries), $\lambda = 4$ (and find up to 4 non-zero entries), \dots and stop when no new non-zero entry is found. The complexity of this approach is, up to a logarithmic factor, the complexity of the last iteration (in which the value of λ is $\lambda = 2^{\lceil \log_2 \ell \rceil + 1}$ if ℓ is a power of two, and $\lambda = 2^{\lceil \log_2 \ell \rceil}$ otherwise). In this paper we will then assume, without loss of generality, that a value λ such that $\ell \leq \lambda \leq 2\ell$ is always available.

3 Finding up to $O(n)$ Non-zero Entries

Let A and B be the two $n \times n$ Boolean matrices of which we want to compute the product. In this section we define, following [11], a graph collision problem and use it to show how to compute up to $O(n)$ non-zero entries of AB .

Let $G = (I, J, E)$ be a bipartite undirected graph over two disjoint sets I and J , each of size n . The edge set E is then a subset of $I \times J$. When there is no ambiguity it will be convenient to write $I = \{1, \dots, n\}$ and $J = \{1, \dots, n\}$. We now define the concept of a collision for the graph G .

Definition 3.1. *For any index $k \in \{1, \dots, n\}$, a k -collision for G is an edge $(i, j) \in E$ such that $A[i, k] = B[k, j] = 1$. A collision for G is an edge $(i, j) \in E$ that is a k -collision for some index $k \in \{1, \dots, n\}$.*

We suppose that the graph G is given by a data structure \mathcal{M} that contains the following information:

- for each vertex u in I , the degree of u ;
- for each vertex u in I , a list of all the vertices of J not connected to u .

The size of \mathcal{M} is at most $\tilde{O}(n^2)$, but the key idea is that its size will be much smaller when G is “close to” a complete bipartite graph. Using adequate data structures to implement \mathcal{M} (e.g., using self-balancing binary search trees), we can perform the following four access operations in $\text{poly}(\log n)$ time.

`get-degree(u)`: get the degree of a vertex $u \in I$

`check-connection(u, v)`: check if the vertices $u \in I$ and $v \in J$ are connected

`get-vert I (r, d)`: get the r -th smallest vertex in I that has degree at most d

`get-vertJ(r, u)`: get the r -th smallest vertex in J not connected to $u \in I$

For the latter two access operations, the order on the vertices refer to the usual order \leq obtained when seeing vertices in I and J as integers in $\{1, \dots, n\}$. We assume that these two access operations output an error message when the query is not well-defined (i.e., when r is too large).

Similarly, we can update \mathcal{M} in $\text{poly}(\log n)$ time to take in consideration the removal of one edge (u, v) from E (i.e., update the degree of u and update the list of vertices not connected to u). This low complexity will be crucial since our main algorithm (in Proposition 3.2 below) will remove successively edges from E .

Let L be an integer such that $0 \leq L \leq n^2$. We will define our graph collision problem, denoted $\text{GRAPH COLLISION}(n, L)$, as the problem of finding a collision for G under the promise that $|E| \geq n^2 - L$, i.e., there are at most L missing edges in G . The formal definition is as follows.

GRAPH COLLISION(n, L) [here $n \geq 1$ and $0 \leq L \leq n^2$]
INPUT: two $n \times n$ Boolean matrices A and B
 a bipartite graph $G = (I \cup J, E)$, with $|I| = |J| = n$, given by \mathcal{M}
 an index $k \in \{1, \dots, n\}$
PROMISE: $|E| \geq n^2 - L$
OUTPUT: one k -collision if such a collision exists

The following proposition shows that there exists a time-efficient quantum algorithm solving this problem. The algorithm is similar to the query-efficient quantum algorithm given in [11], but uses the data structure \mathcal{M} in order to keep the time complexity low.

Proposition 3.1. *There exists a quantum algorithm running in time $\tilde{O}(\sqrt{L} + \sqrt{n})$ that solves, with high probability, the problem $\text{GRAPH COLLISION}(n, L)$.*

Proof. We will say that a vertex $i \in I$ is marked if $A[i, k] = 1$, and that a vertex $j \in J$ is marked if $B[k, j] = 1$. Our goal is thus to find a pair $(i, j) \in E$ of marked vertices. The algorithm is as follows.

We first use the minimum finding quantum algorithm from [6] to find the marked vertex u of largest degree in I , in $\tilde{O}(\sqrt{n})$ time using `get-degree`(\cdot) to obtain the order of a vertex from the data structure \mathcal{M} . Let d denote the degree of u , let I' denote the set of vertices in I with degree at most d , and let S denote the set of vertices in J connected to u . We then search for one marked vertex in S , using Grover's algorithm [10] with `check-connection`(u, \cdot), in $\tilde{O}(\sqrt{n})$ time. If we find one, then this gives us a k -collision and we end the algorithm. Otherwise we proceed as follows. Note that, since each vertex in I' has at most d neighbors, by considering the number of missing edges we obtain:

$$|I'| \cdot (n - d) \leq n^2 - |E| \leq L.$$

Also note that $|J \setminus S| = n - d$. We do a quantum search on $I' \times (J \setminus S)$ to find one pair of connected marked vertices in time $\tilde{O}(\sqrt{|I'| \cdot |J \setminus S|}) = \tilde{O}(\sqrt{L})$, using `get-vertI`(\cdot, d) to access the vertices in I' and `get-vertJ`(\cdot, u) to access the vertices in $J \setminus S$. \square

We now show how an efficient quantum algorithm that computes up to $O(n)$ non-zero entries of the product of two $n \times n$ matrices can be constructed using Proposition 3.1.

Proposition 3.2. *Let λ be a known value such that $\lambda = O(n)$. Then there exists a quantum algorithm that, given any $n \times n$ Boolean matrices A and B such that the number of non-zero entries in the product AB is at most λ , computes AB in time $\tilde{O}(n\sqrt{\lambda})$.*

Proof. Let A and B be two $n \times n$ Boolean matrices such that the product AB has at most λ non-zero entries.

We associate with this matrix multiplication the bipartite graph $G = (V, E)$, where $V = I \cup J$ with $I = J = \{1, \dots, n\}$, and define the edge set as $E = I \times J$. The two components I and J of G are then fully connected: there is no missing edge. It is easy to see that computing the product of A and B is equivalent to computing all the collisions, since a pair (i, j) is a collision if and only if the entry in the i -th row and the j -th column of the product AB is 1.

To find all the collisions, we will basically repeat the following approach: for a given k , search for a new k -collision in G and remove the corresponding edge from E by updating the data structure \mathcal{M} corresponding to G . Since we know that there are at most λ non-zero entries in the matrix product AB , at most λ collisions will be found (and then removed). We are thus precisely interested in finding collisions when $|E| \geq n^2 - \lambda$, i.e., when there are at most λ missing edges in G . We can then use the algorithm of Proposition 3.1. The main subtlety is that we cannot simply try all the indexes k successively since the cost would be too high. Instead, we will search for good indexes in a quantum way, as described in the next paragraph.

We partition the set of potential witnesses $K = \{1, \dots, n\}$ into $m = \max(\lambda, n)$ subsets K_1, \dots, K_m , each of size at most $\lceil n/m \rceil$. Starting with $s = 1$, we repeatedly search for a pair (i, j) that is a k -collision for some $k \in K_s$. This is done by doing a Grover search over K_s that invokes the algorithm of Proposition 3.1. Each time a new collision (i, j) is found (which is a k -collision for some $k \in K_s$), we immediately remove the edge (i, j) from E by updating the data structure \mathcal{M} . When no other collision is found, we move to K_{s+1} . We end the algorithm when the last set K_m has been processed.

This algorithm will find, with high probability, all the collisions in the initial graph, and thus all the non-zero entries of AB . Let us examine its time complexity. We first discuss the complexity of creating the data structure \mathcal{M} (remember that updating \mathcal{M} to take in consideration the removal of one edge from E has polylogarithmic cost). Initially $|E| = n^2$, so each vertex of I has the same degree n . Moreover, for each vertex $u \in I$, there is no vertex in J not connected to u . The cost for creating \mathcal{M} is thus $\tilde{O}(n)$ time. Next, we discuss the cost of the quantum search. Let λ_s denote the number of collisions found when examining the set K_s . Note that the search for collisions (the Grover search that invokes the algorithm of Proposition 3.1) is done $\lambda_s + 1$ times when examining K_s (we need one additional search to decide that there is no other collision). Moreover, we have $\lambda_1 + \dots + \lambda_m \leq \lambda$. The time complexity of the search is thus

$$\tilde{O} \left(\sum_{s=1}^m \sqrt{|K_s|} \times (\sqrt{\lambda} + \sqrt{n}) \times (\lambda_s + 1) \right) = \tilde{O} \left(\sqrt{n}\lambda + n\sqrt{\lambda} \right) = \tilde{O} \left(n\sqrt{\lambda} \right).$$

The overall time complexity of the algorithm is thus $\tilde{O}(n\sqrt{\lambda} + n) = \tilde{O}(n\sqrt{\lambda})$. \square

4 Reduction to Several Matrix Multiplications

Suppose that we have a randomized (or quantum) algorithm \mathcal{A} that, given any $m \times n$ Boolean matrix A and any $n \times m$ Boolean matrix B such that the number of non-zero entries in the product AB is known to be at most L , computes AB with time complexity $T(m, n, L)$. For the sake of simplicity, we will make the following assumptions on \mathcal{A} :

- (1) the time complexity of \mathcal{A} does not exceed $T(m, n, L)$ even if the input matrices do not satisfy the promise (i.e., if there are more than L non-zero entries in the product);
- (2) the algorithm \mathcal{A} never outputs that a zero entry of the product is non-zero;

- (3) if the matrix product has at most L non-zero entries, then with probability at least $1 - 1/n^3$ all these entries are found.

These assumptions can be done without loss of generality when considering quantum algorithms for Boolean matrix multiplication as defined in Section 2. Assumption (1) can be guaranteed simply by supposing that the algorithm systematically stops after $T(m, n, L)$ steps. Assumption (2) can be guaranteed since a witness is output for each potential non-zero entry found (the witness can be used to immediately check the result). Assumption (3) can be guaranteed by repeating the original algorithm (which has success probability at least $2/3$) a logarithmic number of times.

The goal of this section is to show the following proposition.

Proposition 4.1. *Let L be a known value such that $L \geq n$. Then, for any value $r \in \{1, \dots, n\}$, there exists an algorithm that, given any $n \times n$ Boolean matrices A and B such that the number of non-zero entries in the product AB is at most L , uses algorithm \mathcal{A} to compute with high probability the product AB in time*

$$\tilde{O} \left(r^2 \times T \left(\lceil n/r \rceil, n, \frac{100(n + L/r)}{r} \right) + n \right).$$

We will need a lemma in order to prove Proposition 4.1. Let C be an $n \times n$ Boolean matrix with at most L non-zero entries. Let r be a positive integer such that $r \leq n$. We choose an arbitrary partition P_1 of the set of rows of C into r blocks in which each block has size at most $\lceil n/r \rceil$. Similarly, we choose an arbitrary partition P_2 of the set of columns of C into r blocks in which each block has size at most $\lceil n/r \rceil$. These gives a decomposition of the matrix C into r^2 subarrays (each of size at most $\lceil n/r \rceil \times \lceil n/r \rceil$). We would like to say that, since C has L non-zero entries, then each subarray has at most $O(L/r^2)$ non-zero entries. This is of course not true in general, but a similar statement will hold with high probability for a given subarray if we permute the rows and the columns of C randomly. We formalize this idea in the following lemma, which can be seen as an extension of a result proved by Lingas (Lemma 2 in [14]).

Lemma 4.1. *Let C be an $n \times n$ Boolean matrix with at most L non-zero entries. Assume that σ and τ are two permutations of the set $\{1, \dots, n\}$ chosen independently uniformly at random. Let $C[i, j]$ be any non-zero entry of C . Then, with probability at least $9/10$, after permuting the rows according to σ and the columns according to τ , the subarray containing this non-zero entry (i.e., the subarray containing the entry in the $\sigma(i)$ -th row and the $\tau(j)$ -th column) has at most*

$$10 \cdot \left(\frac{L \lceil n/r \rceil^2}{(n-1)^2} + 2 \lceil n/r \rceil - 1 \right)$$

non-zero entries.

Proof. Since the permutations σ and τ are chosen independently uniformly at random, we can consider that the values $\sigma(i)$ and $\tau(j)$ are first chosen, and that only after this choice the $2n - 2$ other values (i.e., $\sigma(i')$ for $i' \neq i$ and $\tau(j')$ for $j' \neq j$) are chosen.

Assume that the values $\sigma(i)$ and $\tau(j)$ have been chosen. Consider the subarray of C containing the entry in the $\sigma(i)$ -th row and the $\tau(j)$ -th column. Let S denote the set of all the entries of the subarray, and let $T \subseteq S$ denote the set of all the entries of the subarray that are in the $\sigma(i)$ -th row or in the $\tau(j)$ -th column. Note that $|S| \leq \lceil n/r \rceil^2$ and $|T| \leq 2 \lceil n/r \rceil - 1$. Let X_s , for each $s \in S$, denote the random variable with value one if the entry s of the subarray is one, and value zero otherwise. The random variable representing the number of non-zero entries in the subarray

is thus $Y = \sum_{s \in S} X_s$. The expectation of Y is

$$\begin{aligned} E[Y] &= \sum_{s \in S} E[X_s] = \sum_{s \in S \setminus T} E[X_s] + \sum_{s \in T} E[X_s] \leq |S \setminus T| \times \frac{L}{(n-1)^2} + |T| \\ &\leq \frac{L \lceil n/r \rceil^2}{(n-1)^2} + 2 \lceil n/r \rceil - 1. \end{aligned}$$

The first inequality is obtained by using the inequality $E[X_s] \leq 1$ for each entry $s \in T$, and by noting that each of the non-zero entries of C that are neither in the i -th row nor in the j -th column has probability exactly $\frac{1}{(n-1)^2}$ to be moved into a given entry in $S \setminus T$ by the permutations of the remaining $n-1$ rows and $n-1$ columns. From Markov's inequality we obtain:

$$\Pr \left[Y \geq 10 \cdot \left(\frac{L \lceil n/r \rceil^2}{(n-1)^2} + 2 \lceil n/r \rceil - 1 \right) \right] \leq \frac{1}{10}.$$

The statement of the lemma follows from the observation that the above inequality holds for any choice of $\sigma(i)$ and $\tau(j)$. \square

Proof of Proposition 4.1. Take two arbitrary partitions P_1, P_2 of the set $\{1, \dots, n\}$ into r blocks in which each block has size at most $\lceil n/r \rceil$. Let us write

$$\Delta = 10 \cdot \left(\frac{L \lceil n/r \rceil^2}{(n-1)^2} + 2 \lceil n/r \rceil - 1 \right).$$

It is easy to show that $\Delta \leq \frac{100(n+L/r)}{r}$ when $n \geq 3$. We will repeat the following procedure $\lceil c \log n \rceil$ times, for some large enough constant c :

1. Permute the rows of A randomly and denote by A^* the resulting matrix;
Permute the columns of B randomly and denote by B^* the resulting matrix;
2. Decompose A^* into r smaller matrices A_1^*, \dots, A_r^* of size at most $\lceil n/r \rceil \times n$ by partitioning the rows of A^* according to P_1 ;
Decompose B^* into r smaller matrices B_1^*, \dots, B_r^* of size at most $n \times \lceil n/r \rceil$ by partitioning the columns of B^* according to P_2 ;
3. For each $s \in \{1, \dots, r\}$ and each $t \in \{1, \dots, r\}$, compute up to Δ non-zero entries of the product $A_s^* B_t^*$ using the algorithm \mathcal{A} .

The time complexity of this procedure is $\tilde{O}(r^2 \times T(\lceil n/r \rceil, n, \Delta) + n)$, where the additive term $\tilde{O}(n)$ represents the time complexity of dealing with the permutations of rows and columns (note that A^*, B^* , the A_s^* 's and the B_t^* 's have not to be computed explicitly; we only need to be able to recover in polylogarithmic time a given entry of these matrices).

We now show the correctness of the algorithm. First, the algorithm will never output that a zero entry of AB is non-zero, from our assumption on the algorithm \mathcal{A} . Thus all the entries output by the algorithm are non-zero entries of AB . The question is whether all the non-zero entries are output.

Let (i, j) be a fixed non-zero entry of AB . Note that each matrix product $A_s^* B_t^*$ corresponds to a subarray of $A^* B^*$. From our assumptions on algorithm \mathcal{A} , this entry will be output with probability $p \geq 1 - 1/n^3$ at Step 3 of the procedure if the entry (after permutation of the rows and the columns) is in a subarray of $A^* B^*$ containing at most Δ non-zero entries. From Lemma 4.1, this happens with probability at least $9/10$. With probability at least $1 - (1/10)^{\lceil c \log n \rceil}$ this

case will happen at least once during the $\lceil c \log n \rceil$ iterations of the procedure. By choosing the constant c large enough, we have $1 - (1/10)^{\lceil c \log n \rceil} \geq 1 - 1/n^3$, and then the algorithm outputs this non-zero entry with probability at least $(1 - 1/n^3)^p \geq 1 - 2/n^3$. By the union bound we conclude that the probability that the algorithm outputs all the non-zero entries of AB is at least $1 - 2/n$. \square

5 Proofs of Theorems 1 and 2

In this section we give the proofs of Theorems 1 and 2.

Proof of Theorem 1. Let A and B be two $n \times n$ Boolean matrices such that the product AB has ℓ non-zero entries. Remember that, as discussed in Section 2, an integer $\lambda \in \{1, \dots, n^2\}$ such that $\ell \leq \lambda \leq 2\ell$ is known.

If $\lambda \leq n$ then the product AB can be computed in time $\tilde{O}(n\sqrt{\lambda}) = \tilde{O}(n\sqrt{\ell})$ by the algorithm of Proposition 3.2. Now consider the case $n \leq \lambda \leq n^2$. By Proposition 4.1 (with the value $r = \lceil \sqrt{\lambda/n} \rceil$), the product of A and B can be computed with complexity $\tilde{O}\left(\frac{\lambda}{n} \times T(n, n, \Delta) + n\right)$, where $\Delta = O(n)$. Combined with Proposition 3.2, this gives a quantum algorithm that computes the product AB in $\tilde{O}(\lambda\sqrt{n}) = \tilde{O}(\ell\sqrt{n})$ time. \square

Proof of Theorem 2. Suppose the existence of a quantum algorithm that computes in time

$$O\left(\lambda\sqrt{n} \cdot n^{-\delta(n)}\right)$$

the product of any two $n \times n$ Boolean matrices such that the number of non-zero entries in their product is at most λ . Let c be a positive constant. Using Proposition 4.1 with the values $r = \lceil cn/\sqrt{\lambda} \rceil$ and $L = n^2$, we obtain a quantum algorithm that computes the product of two $n \times n$ Boolean matrices in time

$$\tilde{O}\left(\frac{n^2}{\lambda} \times T\left(n, n, \frac{100n}{\lceil cn/\sqrt{\lambda} \rceil} + \frac{100n^2}{\lceil cn/\sqrt{\lambda} \rceil^2}\right) + n\right).$$

By choosing the constant c large enough, we can rewrite this upper bound as

$$\tilde{O}\left(\frac{n^2}{\lambda} \times T(n, n, \lambda) + n\right) = \tilde{O}\left(n^{5/2-\delta(n)} + n\right),$$

which concludes the proof of the theorem. \square

Acknowledgments

The author is grateful to Stacey Jeffery, Robin Kothari and Frédéric Magniez for helpful discussions and comments, and for communicating to him preliminary versions of their works. He also acknowledges support from the JSPS and the MEXT, under the grant-in-aids Nos. 24700005, 24106009 and 24240001.

References

- [1] AMOSSEN, R. R., AND PAGH, R. Faster join-projects and sparse matrix multiplications. In *Proceedings of Database Theory - ICDT (2009)*, pp. 121–126.

- [2] BANSAL, N., AND WILLIAMS, R. Regularity lemmas and combinatorial algorithms. In *Proceedings of FOCS'09* (2009), pp. 745–754.
- [3] BUHRMAN, H., AND ŠPALEK, R. Quantum verification of matrix products. In *Proceedings of SODA'06* (2006), pp. 880–889.
- [4] COPPERSMITH, D., AND WINOGRAD, S. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation* 9, 3 (1990), 251–280.
- [5] DOR, D., HALPERIN, S., AND ZWICK, U. All-pairs almost shortest paths. *SIAM Journal on Computing* 29, 5 (2000), 1740–1759.
- [6] DÜRR, C., AND HØYER, P. A quantum algorithm for finding the minimum. arXiv:quant-ph/9607014v2, 1996.
- [7] FISCHER, M. J., AND MEYER, A. R. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory* (1971), pp. 129–131.
- [8] FURMAN, M. E. Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph. *Soviet Mathematics Doklady (English translation)* 11, 5 (1970), 1252.
- [9] GALIL, Z., AND MARGALIT, O. All pairs shortest distances for graphs with small integer length edges. *Information and Computation* 134, 2 (1997), 103–139.
- [10] GROVER, L. K. A fast quantum mechanical algorithm for database search. In *Proceedings of STOC'96* (1996), pp. 212–219.
- [11] JEFFERY, S., KOTHARI, R., AND MAGNIEZ, F. Improving quantum query complexity of Boolean matrix multiplication using graph collision. In *Proceedings of ICALP'12* (2012), pp. 522–532.
- [12] JEFFERY, S., AND MAGNIEZ, F. Improving quantum query complexity of Boolean matrix multiplication using graph collision. arXiv:1112.5855v1, December 2011.
- [13] LE GALL, F. Improved output-sensitive quantum algorithms for Boolean matrix multiplication. In *Proceedings of SODA'12* (2012), pp. 1464–1476.
- [14] LINGAS, A. A fast output-sensitive algorithm for boolean matrix multiplication. *Algorithmica* 61, 1 (2011), 36–50.
- [15] MUNRO, J. I. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters* 1, 2 (1971), 56–58.
- [16] NIELSEN, M., AND CHUANG, I. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
- [17] SEIDEL, R. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences* 51, 3 (1995), 400–403.
- [18] SHOSHAN, A., AND ZWICK, U. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of FOCS'99* (1999), pp. 605–615.

- [19] STOTHERS, A. *On the Complexity of Matrix Multiplication*. PhD thesis, University of Edinburgh, 2010.
- [20] VASSILEVSKA WILLIAMS, V. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of STOC'12* (2012), pp. 887–898.
- [21] VASSILEVSKA WILLIAMS, V., AND WILLIAMS, R. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of FOCS'10* (2010), pp. 645–654.